Example 7.5 Model selection for 2D diffusions via DIC

Etienne A.D. Pienaar

2015

```
rm(list=ls(all=T))
dev.off()
library(DiffusionRgqd)
windows(record=T)
set.seed(200707881)

# Call a simulated dataset into the workspace:
data(SDEsim4)
attach(SDEsim4)

# Have a look at the time series:
plot(Xt~time,type='l',col='blue',ylim=c(0,15),main='Simulated Data',
     xlab='Time (t)',ylab='State',axes=F)
lines(Yt~time,col='red')
axis(1,seq(0,100,5))
axis(1,seq(0,100,5/10),tcl=-0.2,labels=NA)
axis(2,seq(0,15,2))
axis(2,seq(0,15,2/10),tcl=-0.2,labels=NA)

# Define the coefficients of a proposed model:
a00 <- function(t){theta[1]*theta[2]}
a10 <- function(t){-theta[1]}
c11 <- function(t){theta[3]*theta[3]}

b00 <- function(t){theta[4]*theta[5]+theta[7]*sin(0.25*pi*t)}
b01 <- function(t){-theta[4]}
f01 <- function(t){theta[6]*theta[6]}

# Place some prior distributions on theta[1] and theta[4]:
priors <- function(theta){dnorm(theta[1],1,5)*dnorm(theta[4],1,5)}

# Some starting parameters for the MCMC procedure:
mesh    <- 10                       # Number of mesh points per transition.
updates <- 100000                   # Number of updates for each chain.
burns   <- 50000                    # Number of discarded updates.
X <- cbind(Xt,Yt)                   # Combine the data into a column matrix .
th <- c(5,5,5,5,5,5,5)             # Some starting values for the parameters.

# Define some proposal std. devs. and run the MH-algorithm:
par.sds <- c(0.22,0.30,0.02,0.11,0.04,0.01,0.21)
m1 <- BiGQD.mcmc(X,time,mesh,th,par.sds,updates,burns)

# Calculate parameter estimates:
GQD.estimates(m1,thin=200)

# Run the MCMC procedure M=4 times for each model:
M  <- 4
```

```
SaveOutput  <- list() # Create some  storage for model objects.
M.counter   <- 1      # Create a count variable.
Tot.counter <- 1      # An overall count variable.
kill.count  <- 1      # Set fail-over counter.

while((M.counter<=M)&(kill.count<20))
{
  # Make a new run from a random starting point each time:
  th <- runif(7,1,10)
  m1 <- BiGQD.mcmc(X,time,mesh,th,par.sds,updates,burns,
                   Tag=paste0('Model_A_run_',M.counter))

  # If the chain does not fail, store the output. If the chain fails,
  # keep track of the number of failures:
  if(!m1$failed.chain)
  {
    SaveOutput[[Tot.counter]]  <-  m1
    Tot.counter <- Tot.counter+1
    M.counter   <- M.counter +1
    kill.count  <- 1
  }else
  {
    kill.count <- kill.count+1
  }
}

GQD.remove()
a00 <- function(t){theta[1]*theta[2]}
a10 <- function(t){-theta[1]}
c00 <- function(t){theta[7]*(1+sin(0.25*pi*t))}
c10 <- function(t){theta[3]*theta[3]}

b00 <- function(t){theta[4]*theta[5]+theta[8]*sin(0.25*pi*t)}
b01 <- function(t){-theta[4]}
f01 <- function(t){theta[6]*theta[6]}

priors <- function(theta){dnorm(theta[1],1,5)*dnorm(theta[4],1,5)}

par.sds    <- c(0.17,0.29,0.07,0.10,0.04,0.01,0.83,0.19)
M.counter  <- 1 # Reset the counter.
kill.count <- 1 # Reset the error counter.
while((M.counter<=M)&(kill.count<20))
{
  th <- runif(8,1,10)
  m2 <- BiGQD.mcmc(X,time,mesh,th,par.sds,updates,burns,
                   Tag=paste0('Model_B_run_',M.counter))

  if(!m2$failed.chain)
  {
    SaveOutput[[Tot.counter]]  <- m2
    Tot.counter <- Tot.counter+1
    M.counter   <- M.counter +1
    kill.count  <- 1
```

```
  }else
  {
    kill.count <- kill.count+1
  }
}

GQD.remove()
a00 <- function(t){theta[1]*theta[2]}
a10 <- function(t){-theta[1]}
a01 <- function(t){theta[7]}
c11 <- function(t){theta[3]*theta[3]}

b00 <- function(t){theta[4]*theta[5]+theta[8]*sin(0.25*pi*t)}
b01 <- function(t){-theta[4]}
f01 <- function(t){theta[6]*theta[6]}

priors <- function(theta){dnorm(theta[1],1,5)*dnorm(theta[4],1,5)}

par.sds  <- c(0.18,0.95,0.02,0.10,0.03,0.01,0.23,0.18)
M.counter  <- 1 # Reset the counter.
kill.count <- 1 # Reset the error counter.

while((M.counter<=M)&(kill.count<20))
{
  th <- runif(8,1,10)
  m3 <- BiGQD.mcmc(X,time,mesh,th,par.sds,updates,burns,
                   Tag=paste0('Model_C_run_',M.counter))
  if(!m3$failed.chain)
  {
    SaveOutput[[Tot.counter]]  <-  m3
    Tot.counter <- Tot.counter+1
    M.counter   <- M.counter +1
    kill.count  <- 1
  }else
  {
    kill.count <- kill.count+1
  }
}

GQD.dic(SaveOutput)
```